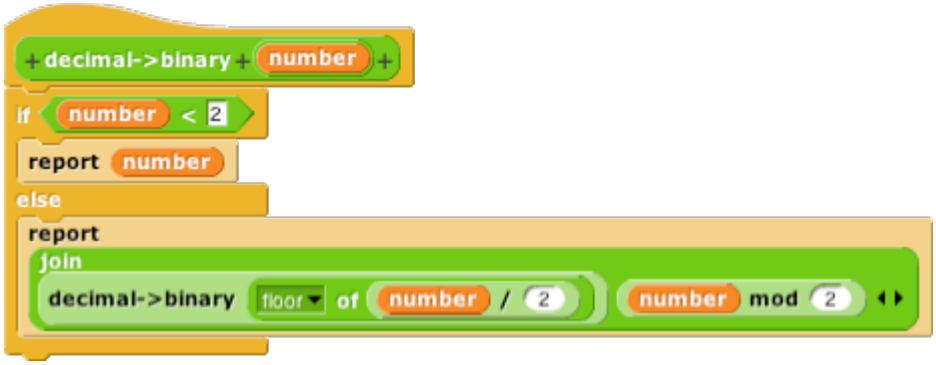
Beyond Binary

Here's our solution:

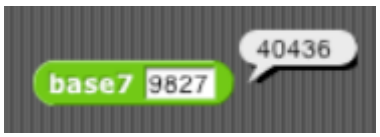


The base case is that the number fits in a single bit—that is, it has to be less than 2. If so, the number itself, 0 or 1, is the desired output.

In the recursive case, the rightmost bit of the result is the remainder of dividing the number by 2. That is, even numbers end with 0, and odd numbers end with 1. The rest of the result is a recursive call on the (integer) quotient of the number divided by 2. The combiner is `join` because we want to string the digits together. It may be surprising that we don't use an arithmetic operator, since we're working with numbers, but the desired result is a ***numeral,*** which is a visible representation of a number, rather than the numeric value itself. A numeral is a text string, so the combiner is a string operation.

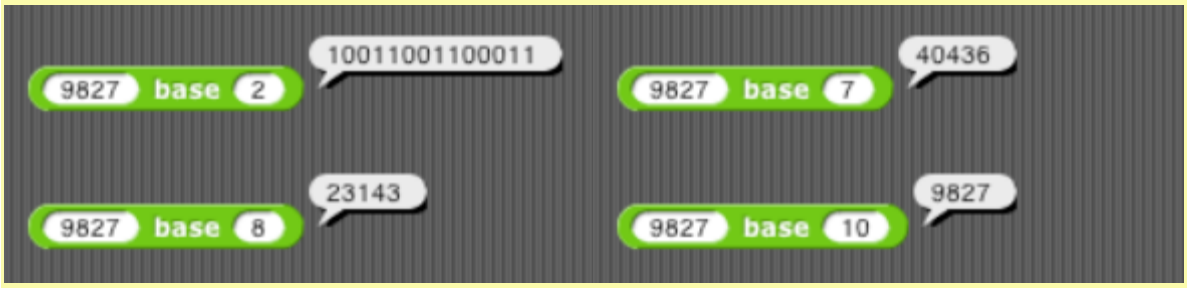
Other Bases

There's no reason to limit ourselves to decimal (base 10) and binary (base 2). How about base 7?

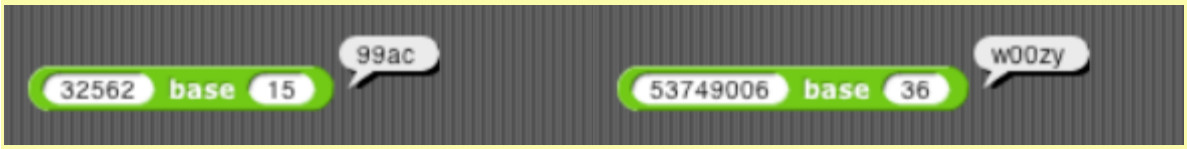


The base 7 digits are 0–6, and the digit positions represent powers of 7.

☞ Write the `base7` block. Then generalize the pattern with a `base` block that takes the base as a second input:



☞ Optional for hotshots: Improve the `base` block so that it can go up to base 36 by using the letters `a`–`z` as digits with values 10–35.



☞ Write the inverse function `from base` that takes a (text) string of digits and a base as inputs, and reports the corresponding number (which Snap***!*** will show in decimal, of course, but you're converting to a***number,*** not to a string of digits). Again, bases above ten are optional.